Journal of Engineering Science and Technology
Vol. 12, No. 1 (2017) 098 - 112

SYMMETRIC ENCRYPTION USING PRE-SHARED PUBLIC PARAMETERS FOR A SECURE TFTP PROTOCOL

N. N. MOHAMED*, Y. M. YUSSOFF, M. A. M. ISA, H. HASHIM

Faculty of Electrical Engineering, University Teknologi MARA,
Shah Alam, Selangor, 40450 Malaysia
*Corresponding Author: nurnabilamohamed@gmail.com

Abstract

Advances in the communication technology of embedded systems have led to the situation where nowadays almost all systems should implement security for data safety. Trivial File Transfer Protocol (TFTP) has advantages for use in embedded systems due to its speed and simplicity, however without security mechanisms, it is vulnerable to various attacks. As an example, during upgrading of Wireless Access Points (WAPs), attackers can access the information and modify it, and then install malicious code to interrupt the system. This work proposes security implementation of Diffie Hellman Key Exchange in TFTP by pre-sharing public parameters that enable two parties to achieve same secret key without the risk of Man-In-The-Middle (MITM) attacks. The implementation is integrated with compression and encryption methods to significantly reduce computational requirements in TFTP communication.

Keywords: Trivial file transfer protocol, AES encryption, Diffie Hellman key exchange, SHA2, Huffman coding.

1. Introduction

TFTP was designed for configuring/transferring files, updating WAPs and other machines that may not have sufficient storage devices. Due to its simple implementation, this protocol is used frequently on constrained embedded devices. However, the main problem in this protocol is that it does not provide any security mechanisms, making it vulnerable to various attacks especially on WAPs [1,2]. It is difficult to ignore the security issues because in such situations, attackers can tamper a firmware or non-volatile memory on the WAP and install malicious codes to destroy the whole system. Another issue is the limitation of the

Nomenclatures

A	Alice's public value, known to Alice, Bob and Eve
a	Alice's secret value, known only to Alice
B	Bob's public value, known to Alice, Bob and Eve
b	Bob's secret value, known only to Bob
g	Primitive root modulo p, known to Alice, Bob and Eve
IA	Alice's secret key which is equivalent to Bob's
IB	Bob's secret key which is equivalent to Alice's
p	Large prime numbers, known to Alice, Bob and Eve

Abbreviations

AES	Advanced Encryption Standard
DAA	Digest Access Authentication
DES	Data Encryption Standard
DHKE	Diffie Hellman Key Exchange
EAP	Extensible Authentication Protocol
FTP	File Transfer Protocol
IPSec	Internet Protocol Security
MITM	Man-In-The-Middle
NIST	National Institute of Standards and Technology
SFTP	Secure File Transfer Protocol
SHA	Secure Hash Algorithm
SSH	Secure Shell
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
WAPs	Wireless Access Points
WPA1	Wi-Fi Protected Access 1
WPA2	Wi-Fi Protected Access 2

data packets which can be sent through normal protocol should not be more than 32 MB [3] thus this constraint keeps its implementation limited, allowing only conventional operations.

In order to strengthen the communication protocol during remote update of firmware in WAP using TFTP, this work presents the implementation of Diffie Hellman Key Exchange (DHKE) by pre-sharing public parameters that enables two communicating parties to achieve the same secret key. The significance of applying pre-shared technique is that it can prevent MITM attacks. The key obtained is then utilized for encrypting/decrypting data using symmetric encryption scheme. For protecting the integrity of data and files, this work also proposed cryptographic hash function for integrity checks. Other than security issue, we also considered compression technique to address the need to reduce information size in TFTP communication. An experiment was carried out to test the security scheme using two embedded devices to present the functionality of key exchange concept and data encryption in TFTP. The results were analysed in terms of execution time, file scheme throughput, average of file reduction space percentage and total execution time using variable file size. This research work is intended to carry out the idea of implementing the pre-shared parameters

technique for securing TFTP communication to be used in real time applications on constrained embedded devices.

2. Literature Review

2.1. Constraints In TFTP

TFTP is generally used for booting system device and upgrading router or firmware due to its simple design. However, it does not support any authentication or encryption mechanism which makes it vulnerable to various attacks. It is impractical to use another advanced protocol in this condition such as File Transfer Protocol (FTP) and Secure File Transfer Protocol (SFTP), due to the complex commands and the use of Transmission Control Protocol/Internet Protocol (TCP/IP) for the transport protocol and for connection. The security issue in TFTP should not be ignored as attackers can easily access and modify the secret information and install malicious codes to interrupt the system. It also has been stated in RFC 3617 [4] that TFTP has no mechanism for access control within the protocol, and there is no protection from MITM attack. Consequently, this become a major security threat in TFTP application especially during updating router or wireless WAP and during remote system update.

Based on previous work in [1][2], it is found that attacks on WAP during upgrading firmware using TFTP enable malicious users to tamper and access the non-volatile memory in the WAP, and then insert malicious codes such as a backdoor in the boot loader, kernel or application. Such attacks allow hackers to modify the private data of any clients that are connected to the infected AP. Existing security solutions such as Wi-Fi Protected Access 1 (WPA1), Wi-Fi Protected Access 2 (WPA2), Extensible Authentication Protocol (EAP) etc are not strong enough to detect security problem and protect the AP from this attack. The author in [5] also claims that TFTP is dangerous to be used in embedded system because there is no security implementation. Besides, as mentioned earlier, TFTP can send 0-512 bytes of data packet at a time except for final block of the transfer. There is no issue when sending small file size [3]. However, the problem arises when sending large packet size as the total of data to transfer sums up to 32 MB which means TFTP is unable to manage files larger than 32MB in standard mode. Thus the obvious solution to overcome the large file transfer issue is by implementing data compression algorithm.

2.2. Diffie Hellman Key Exchange Protocol

The Diffie Hellman key exchange algorithm was developed to provide a secure solution for exchanging information publicly. Several studies have been made to the implementation of DHKE method that allows two parties which have no prior knowledge of each other to jointly establish a shared secret key over the public communication channel [6]. According to E. Yoon [7], this method enables two sides to generate the same private cryptographic key without transferring the key in physical manner. The significance of employing DHKE is that it allows users to securely exchange secret keys in public communication channel. In order to explain the concept in a less complicated way, the scenario on communication between Alice and Bob to exchange secret data over the public channel is

explained. Eve, an intruder who wants to obtain data when Alice and Bob change any data with each other is always monitoring their communication as well. Table of nomenclatures below lists the parameters participate in DHKE agreement.

Based on equation (1), Alice and Bob agreed on g and p value over the internet. Alice chooses a as her private value, and computes the public value A,

$$A = g^{a} \ mod \ p \tag{1}$$

Alice then sends the value A to Bob. Bob on the other hand chooses b as his private value and computes the public value B,

$$B = g^{b} \ mod \ p \tag{2}$$

and sends the value B to Alice. After exchanging the values, Alice and Bob will calculate the value for the key respectively using equation (3) and (4) as follows:

$$IA = B^{a} \ mod \ p \tag{3}$$

$$IB = A^{b} \ mod \ p \tag{4}$$

Based on the above equations, the value for IA and IB should be the same where that value is the secret key. To prove that the value is same, the equation in (3) can be derived as

$$IA = (g^{b})^{a} \bmod p \tag{5}$$

$$= g^{ba} \ mod \ p \tag{6}$$

The equation in (4) also can be derived as

$$IB = (g^{a})^{b} \ mod \ p \tag{7}$$

$$= g^{ab} \ mod \ p \tag{8}$$

IA and IB is compared after the derivation,

$$IA = IB \tag{9}$$

$$g^{ba} \ mod \ p = g^{ab} \ mod \ p \tag{10}$$

Therefore it is proven that the final value for IA and IB is similar. The eavesdropper, Eve, who has been monitoring the communication has a chance to collect the values for g, p, A and B which is shared publicly.

2.3. Man-In-The-Middle Attack (MITM)

The limitation of DHKE is that it is unable to serve authentication property to the communication parties, and it is vulnerable to MITM attack where the adversary can intercept all the parameters passing through the public channel. According to the DHKE protocol, when Alice computes A using a and sends to Bob, Eve as MITM, intercepts by sending her own public value. Eve will also send her own public value A' to Alice. Eve and Alice agree on their shared key whereas Eve and Bob also agree on another shared key. Once it is done, she can decrypt the

symmetric key which is exchanged between Alice and Bob and compromise the entire communication. Thus, in this work, the application of DHKE concept by pre-sharing several parameters to achieve secret key can mitigate MITM attack.

2.4. AES Symmetric Encryption

Popular symmetric key algorithms that have been used to protect data are Data Encryption Standard (DES) and Advanced Encryption Standard (AES). However, in 1997, AES was designed to replace DES due to many attacks and security vulnerability [8]. This problem has occurred because DES has a smaller key size which makes it vulnerable and can be hacked using brute force methods, AES algorithm was selected for several reasons. Most studies in AES found that this algorithm is fast in both software and hardware, therefore it is efficient in wide range of platforms. A comparative study between DES, 3DES and AES among nine criteria has been analysed in [9]. This study has proven that AES is better than DES and 3DES due to its larger key size. Besides, the comparison between block cipher (AES) and stream cipher (RC4) in [10][11] concluded that it is more efficient to encrypt smaller packet data using AES while stream cipher, RC4 is more efficient in encrypting larger data size. Nevertheless, Fluhrer and many researchers have discovered several vulnerabilities of using stream cipher RC4 algorithm [12].

2.5. Cryptographic Hash Function

The most widely used hash function nowadays is Secure Hash Algorithm (SHA) [13] that was introduced publicly in 1993. The first version, 160 bits hash function called SHA-0 had serious security issue due to unrevealed significant flaw and has been replaced by revised version, SHA-1 for information security such as for Secure Socket Layer (SSL), Internet Protocol Security (IPSec) and Secure Shell (SSH). In 2001, an update version, SHA-2 was developed by National Institute of Standards and Technology (NIST), using 256 and 512 bit for general use, nevertheless it has weakness in low processing speed. When using symmetric encryption such as AES to protect data, it definitely ensures confidentiality of message but not the integrity. Hence, implementing cryptographic hash such as SHA-2 in this work will ensure the message is not being altered or changed during the transmission.

2.6. Compression and Encryption Approach

Implementation of compression algorithms universally also produces a fatal security hole. Therefore, existing encryption algorithm which is compatible to be integrated with compression coding such as AES, DES and so on can be used to protect the compressed data. The author in his paper [14] mentioned that symmetric encryption is one of the best methods to protect compressed data. Research in [15] stated that it is recommended to execute compression technique first followed by encryption as an intruder has less cleave to access data when using this sequence. In the paper [16], the compression technique is performed using two types of lossless compression algorithms followed by encrypting data using symmetric encryption, DES however there is no explanation on the secret

key exchange between those systems. The work done in [17] also utilized symmetric encryption, DES to protect compressed data.

2.7. Related Works In Secure TFTP

The intention on providing security mechanism in TFTP is to prevent from various attacks to be used in embedded systems of low computational power. So far, however, most studies in securing TFTP have been carried out in very small number of areas. Recent work in 2013 [18] proposed a security extension to the existing TFTP protocol utilizing Digest Access Authentication (DAA) followed by hash function SHA1 for data authentication. In this work, a random value called nonce is generated and hashed using SHA1 which is then sent to the host device for username and password authentication. This work also proposed two encryption schemes to establish data confidentiality, XTEA and AES as these method are considered secure for all known attacks. However, the results on generating nonce value have not been discussed by the authors of [18], and also there is no proof that this method is considered secure as well.

On the other hand, the framework for security enhancement in TFTP was proposed in 2012 [19]. In order to provide security negotiation such as attestation process before file transfer occur, the author proposed a new packet header using existing TFTP option extension showed in Figure 1. To represent the security protocol, the filename field contains the name of the file string to be read, "TCG_ATTESTATION". The mode field contains one of the strings; netascii, octet or mail. "CRYPTO_ALGO_KEY" string that is used for key exchange protocol is replaced in opt1 and DHKE is used as input in value1 field for simple implementation. The opt2 is placed with "CRYPTO_ALGO_DATA" string that is used for data encryption of all data packets. AES is chosen based on common data encryption as input in value2 field. The option negotiation mechanism proposed is a backward-compatible extension to the TFTP protocol. Nevertheless, its implementation and integrity issue have not been discussed to ensure the encrypted data payload is unchanged by adversary.

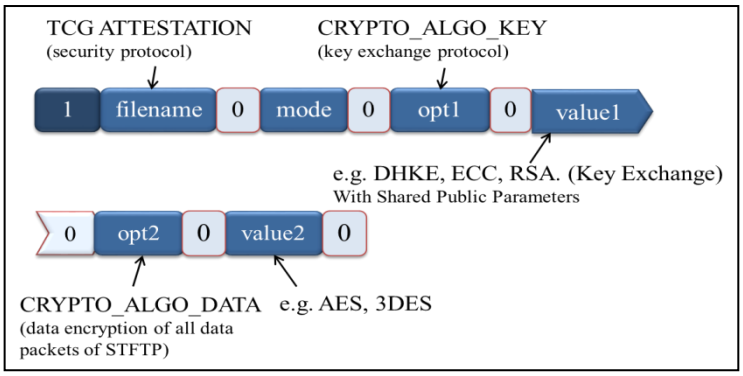

Fig. 1. TFTP Option Extension RRQ Packet [19].

The ease of implementation is the main reason of using TFTP in embedded system nevertheless the lack of security mechanism becomes the significant problem. The advantages of proposed work done in [19] using a simple symmetric encryption (AES algorithm) and asymmetric principle (DHKE

concept) can perform considerable security enhancement in TFTP communication. The continuance work done in [16] implies lossless data compression followed by data protection utilizing symmetric cryptography to optimize secured communication between two wireless entities. Consequently, these two scholars are selected as the fundamental for this research.

3. Proposed Methodology

This work is done using two embedded microprocessors, Raspberry Pi ARM Boards acted as client and server where those devices are placed in a distance about a meter. Both devices are associated with keyboard and mouse to execute program and a monitor to display the program. For power adapter, the micro usb power supply which provides at least 700mA at 5V is essential. In this work, PC power supply is used to power up the device. Each workstation is communicating via wireless connection using wireless tools called Wi-Pi dongle. Apart from hardware configuration, additional software is also installed on each raspberry pi. The OS Linux Raspbian 6 (Wheezy) is installed on each workstation in SD card 2GB attached on the board to facilitate file transmission process through TFTP as well as implementation of compression and encryption algorithm.

In this work, the pre-sharing of public parameters between client and server (*p*, *g* and *A*) is proposed. This is established during initial communication and is assumed to have been completed in the subsequent discussion. Figure 2 illustrates the proposed DHKE concept for both client and server share the public parameters in TFTP communication. The client begins the communication by sending Read Request (RRQ) packet to obtain server's public value, *B*. The RRQ packet sent contains opcode 1 and the filename field contains RRQ of Public Key, B. After acknowledging the request, the server sends DATA packet containing a 3 in the opcode field, the block# starts with 1 and DATA field contains value *B*. The client acknowledges the transmission by sending Acknowledgment (ACK) packet containing opcode 4 and block# 1. A payload of less than 512 bytes will finally end the file transfer. It is important that the public value *B* should always be different in every transmission to prevent adversary compute the key easily. Thus, /dev/random that generates random number is utilized to obtain random value. Besides, prime value *p* should also large enough not to be guessed, to defend against attack.

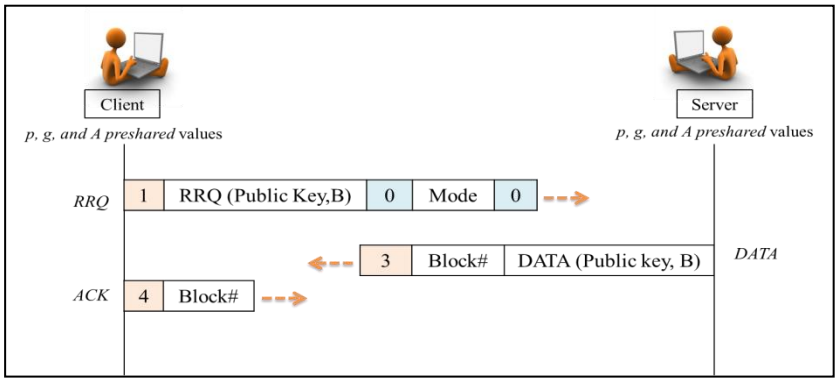

Fig. 2. Pre-shared Public Parameters in TFTP.

Once the public parameters exchange is done and the secret key for symmetric encryption and decryption is obtained, the security mechanism to protect data is implemented. Also, the compression technique is performed to address the need for time saving during transmission. Based on the literature review, it is recommended to perform compression first followed by encryption especially when utilizing text data [15]. Thus in this work, the server compresses the plaintext file using Huffman coding algorithm and encrypt the file using the selected methods; AES symmetric encryption algorithm for securing file content and SHA2 cryptographic hash function to generate hash value for correct message identification. Compressed/encrypted file is sent to the client side through TFTP. Since the hash value is not encrypted, it is therefore sent separately from the encrypted file. After transmission, at the client side, hash value is compared to make sure there is no modification and is decrypted using AES algorithm. Finally, the decrypted file is decompressed using the same Huffman coding algorithm to retrieve the original file. Figure 3 shows overall process of sending file with compression and encryption scheme through TFTP RRQ packet transmission.

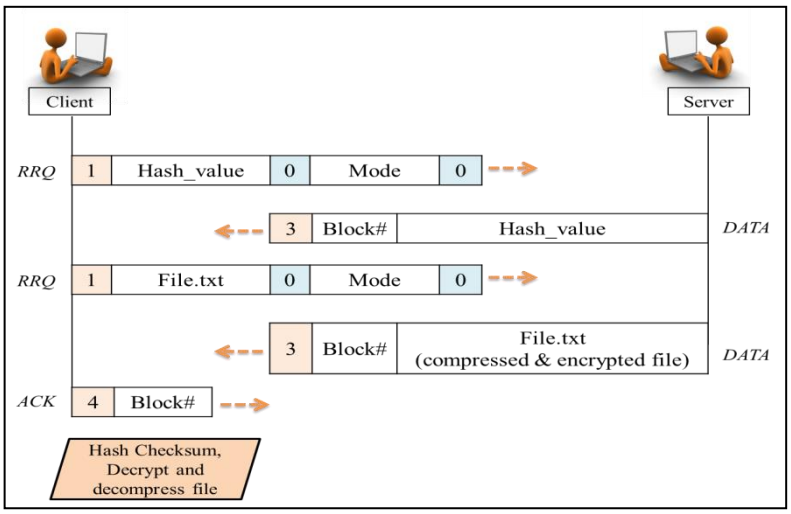

Fig. 3. TFTP With Security Component.

3.1. Compression and Security Performance

This section explains the method to evaluate the effectiveness and performance of compression and security scheme used. There are several file schemes criteria on measuring the performance in this work as follows:

- Original file is the plaintext file without any modification including encryption or compression scheme. All files are in text format (.txt).
- File compressed/decompressed using Huffman coding algorithm (compress only and decompress only).
- File encrypted/decrypted using AES encryption algorithm (encrypt only and decrypt only).
- File compressed/encrypted and decompressed/decrypted using Huffman coding, AES encryption and SHA2 hash function.

3.1.1. Checking Both Sides Get Same Secret Key

After both sides obtain their public values, the client computes the key based on the equation, $key = B^a \bmod p$ meanwhile server computes its key using the equation $key = A^b \bmod p$ where both key value will turn out to be the same. In order to prove both sides obtain the same shared secret, a test program will be used using C programming.

3.1.2. Execution Time of Compression/Encryption Scheme

The execution time of compression/encryption scheme, and also decryption/decompression scheme is the time consumption of methods applied in this work. If the execution time of the technique is less or in an acceptable level compared to the time of independent common encryption and compression, it indicates that the algorithm is acceptable with the time factor respectively.

3.1.3. File Scheme Throughput

File scheme throughput defines the speed of compression, encryption and hashing scheme. From the result of execution time, the throughput of every scheme is calculated to indicate the speed as in below equation:

$$Throughput = \frac{TotalFileSize(kB)}{ExecutionTimeof\ EachScheme} \tag{11}$$

3.1.4. Average File Size Reduction Space

The average of file size reduction space calculates the shrinkage of the source file as a percentage using equation below.

$$Average = \frac{Size\ before\ compression/encryption - Size\ after\ compression/encryption}{Size\ before\ compression/encryption}\% \tag{12}$$

3.1.5. Total Execution Time

Total execution time in TFTP is the total time consumed for compressing/encrypting process include for the transmission time to send requested file from the server to the client side, also sum up with the decompressing/decrypting process.

4. Results and Discussion

4.1. Mitigation of MITM Attack

Client computes the key based on the equation, $key = B^a \bmod p$ meanwhile server computes its key using the equation $key = A^b \bmod p$ where both key value turns out to be the same. The shared secret key for both sides is similar based on the equation $key = g^{ab} \bmod p$. The client and server now have the same shared secret and even if someone tried to listen or modify the value on the untrusted channel, it is impossible for them to compute the secret key from the

captured information, as computing a discrete logarithm of each public value is practically unfeasible.

MITM attack arises as DHKE does not authenticate the communicating participants. This attack enables an opponent to intercept the communication by substituting his/her own public value to both communicating parties. As an example, when Bob transmits his public value to Alice, Eve intercepts it by sending her own value to Alice. Eve and Alice thus agree on one shared key as well as Eve and Bob agree on another shared key. After this exchange, Eve can simply use the key to decrypt any messages sent out by Alice or Bob, and possibly modifies them before transmitting to the other side. Implementation of the pre-shared public parameters has made it possible for DHKE scheme to be safe from MITM attack as only the server side sends the public value to the other side. As shown in Figure 4, the intruder may obtain and modify the value of server's public which is B however; since both of the communicating parties already have knowledge of one of the public values (A), it will not compromise the communication. Therefore, the proposed technique is considered secure from MITM attack because the preinstalled public value cannot be accessed by adversary thus this becomes the strength in this protocol.

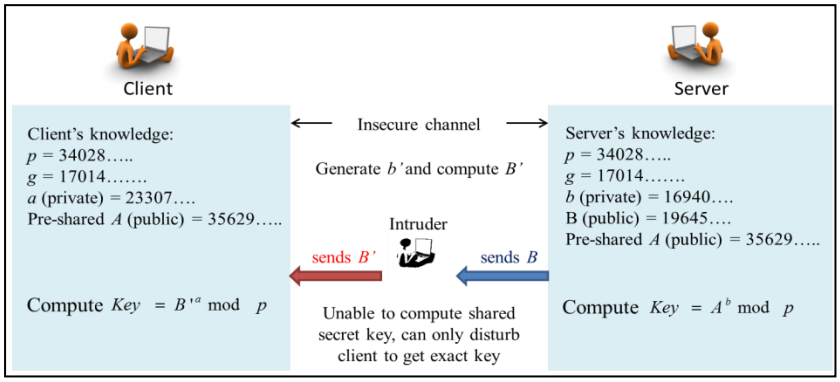

Fig. 4. Pre-shared Public Parameters in DHKE.

4.2. Experimental Result for Execution Time to Compress, Encrypt and Compress/Encrypt File and Discussion

According to the results on Table 1, processing time when using the three methods increase as the file size increase. The average total time for compress only was 0.186 seconds whereas the average total time for encrypt only scheme was 0.274 seconds. Meanwhile, the average total time for combined compress/encrypt scheme was 0.333 seconds which indicates that this scheme is slow in producing the output file compared to using compress only and encrypt only algorithm, however it has merit on file protection as it implements security mechanism compared to compression alone.

Table 1. Result on Time for Compress Only, Encrypt Only, and Compress/Encrypt Scheme.

File size (kB)	Compress Only Time(s)	Compress Only Throughput (kB/s)	Encrypt Only Time(s)	Encrypt Only Throughput (kB/s)	Compress/ Encrypt Time (s)	Compress/ Encrypt Throughput (kB/s)
9.22	0.012	768.33	0.001	9220.00	0.012	768.33
51.10	0.016	3193.94	0.004	12775.75	0.016	3193.94
102.21	0.012	8517.33	0.008	12776.00	0.024	4258.67
511.05	0.028	18251.71	0.028	18251.71	0.040	12776.20
1024.00	0.036	28444.44	0.044	23272.73	0.060	17066.67
2047.45	0.064	31991.47	0.080	25593.18	0.112	18280.84
5119.39	0.140	36567.06	0.200	25596.95	0.248	20642.70
10239.80	0.264	38787.14	0.396	25858.09	0.476	21512.19
20479.61	0.516	39689.16	0.784	26121.95	0.932	21973.83
30719.41	0.768	39999.23	1.192	25771.32	1.408	21817.76
Average Time (s)	0.186		0.274		0.333	

In this work, the throughput shown in Figure 4 is the speed of the compression and encryption scheme. Although the throughput of proposed compression/encryption method is the lowest, there is only slight difference with encrypt only time, as well as it has advantage in terms of reducing file size. Besides, compared to compress only, the proposed scheme is definitely more secure. Moreover, this technique applies cryptographic hash to ensure data has not been modified during transmission.

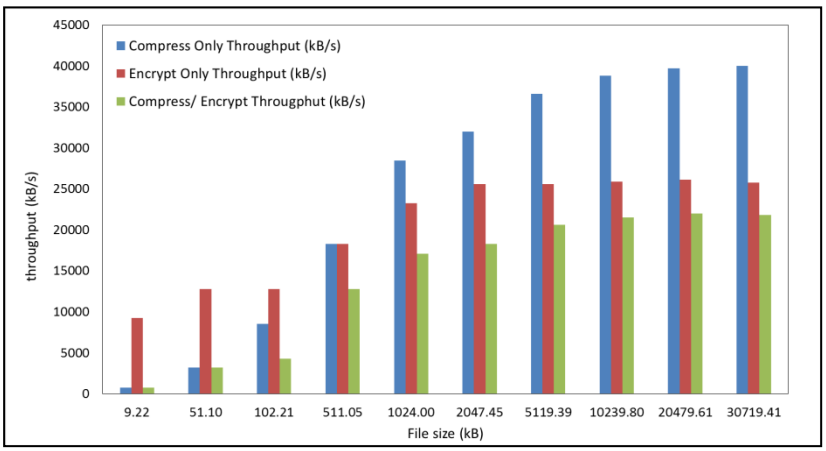

Fig. 5. Throughput to Compress Only, Encrypt Only and Compress/Encrypt File.

4.3. Experimental Result for Execution Time to Decompress, Decrypt, and Decrypt/Decompress File and Discussion

Based on Table 2, the execution time to decompress and decrypt file after file transmission are also compared. Similar to previous result, for all three methods, execution time increases as the file size increases. It takes the average about 0.179

seconds for decrypt only time while 0.127 seconds for decompress only time. The average execution time for decryption/decompression is 0.192 seconds. Although it takes the longest time when using the combined scheme, a difference of only 0.013 seconds was observed when compared to decrypt only time.

Table 2. Result on Time for Decompress Only, Decrypt Only, and Decrypt/Decompress Scheme.

File size (kB)	Decrypt Only Time (s)	Decrypt Only Throughput (kB/s)	Decompress Only Time (s)	Decompress Only Throughput (kB/s)	Decrypt/ Decompress Time (s)	Decrypt/ Decompress Throughput (kB/s)
9.22	0.008	1152.50	0.002	4610.00	0.012	768.33
51.10	0.012	4258.58	0.003	17034.33	0.008	6387.88
102.21	0.012	8517.33	0.004	25552.00	0.012	8517.33
511.05	0.028	18251.71	0.012	42587.33	0.028	18251.71
1024.00	0.032	32000.00	0.020	51200.00	0.040	25600.00
2047.45	0.060	34124.23	0.044	46533.05	0.068	30109.62
5119.39	0.136	37642.57	0.092	55645.53	0.150	34129.26
10239.80	0.228	44911.42	0.184	55651.11	0.272	37646.34
20479.61	0.498	41123.71	0.364	56262.66	0.520	39383.86
30719.41	0.774	39689.16	0.540	56887.80	0.808	38019.07
Average Time (s)	0.179		0.127		0.192	

Figure 5 below shows the result of the throughput plotted in the graph. The throughput using the combined scheme was also found to be slower than the other two independent scheme and showed not much difference with the previous throughput result.

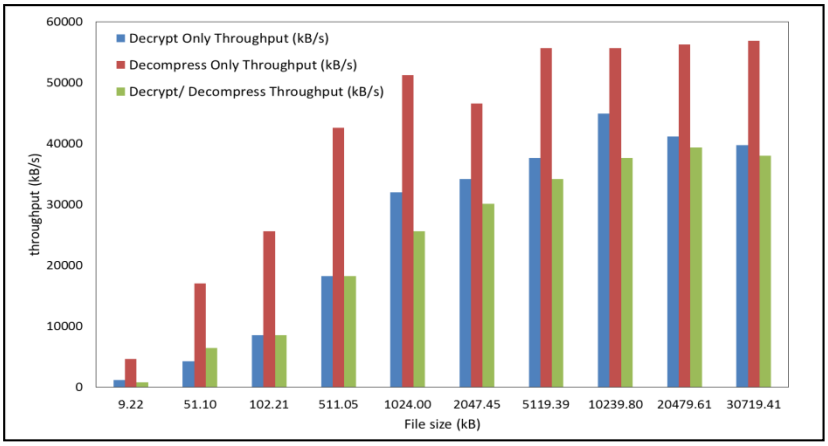

Fig. 6. Throughput to Decompress, Decrypt and Decrypt/Decompress File.

4.4. Average Percentage of File Reduction Space

The average of file size reduction space is evaluated to show the percentage of shrinkage of the source file shown in Table 3. Lowest saving percentage is given by encrypt only technique using AES algorithm which is about the average of

-0.5%. Meanwhile the average values of both compress only technique using Huffman coding algorithm, and combined compression/encryption scheme using Huffman compression and AES encryption algorithm are almost similar which is about 70-71% reduction from the original source file; the differences do not exceed more than 1%. However, compression only method focuses on compression without data protection which enable unauthorized person to read the content. This indicates that combination scheme using Huffman compression and AES encryption algorithm shows a great space saving thus this is the best scheme as it reduces data volume and ensures security.

Table 3. Average Percentage of File Reduction Space.

Compress Only (%)	Encrypt Only (%)	Compression-Encryption (%)
70.9219	-3.3080	67.6681
71.1602	-1.6320	70.5595
71.1862	-0.2867	70.8937
71.2062	-0.0589	71.1485
71.2088	-0.0286	71.1792
71.2100	-0.0144	71.1952
71.2108	-0.0058	71.2049
71.2111	-0.0029	71.2082
71.2112	-0.0015	71.2097
71.2113	-0.0010	71.2103

4.5. Total Execution Time

Figure 7 shows that the total execution time for compress only scheme is the fastest among the other schemes, however, the time is similar with the proposed combined scheme. When compared with the original or normal file, the proposed scheme was able to reduce execution time by about 30%. This can be considered as a significant value along with the fact that it combines both schemes to minimize time and to secure information. While for the encrypted file, it has increased few bytes after encryption, thus the execution time is the slowest.

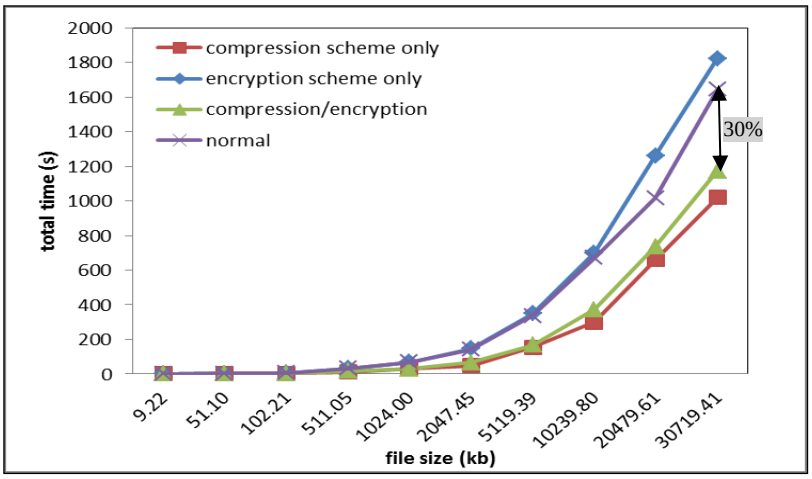

Fig. 7. Total Execution Time.

5. Conclusions

This work presents DHKE concept to exchange public values between two communicating parties using TFTP with pre-shared public parameters. The idea of pre-shared technique in which only one side send the public value concept can prevent at least MITM attack becomes the strength in TFTP protocol secure communication, as the client and server has already been known using several parameters to compute the secret key. This work is proposed as a preliminary work to test the idea for implementation on energy constrained device and to support future research on securing embedded device. The concept is integrated with compression and encryption technique to significantly reduce the computational requirements in TFTP communication. For key exchange, it has been proven in the experiment that both client and server obtained the similar value for the key that is used for encryption and decryption. However, the weakness in this work is the adversary may act as a server. Therefore, the solution such as using session token can prevent this type of attack.

On the other hand, implementation of compression and encryption scheme gives good compression ratio as well as offers great file space saving thus making it the best scheme among the three schemes studied as it minimizes data volume and provides data security. We have considered cryptographic security implementation in TFTP utilizing at least DHKE for key exchange, AES for protecting data confidentiality and also cryptographic hash function SHA2 to protect information from being tampered or eavesdropped by unauthorized third parties. The major contribution of this work is it implements DHKE concepts through pre-sharing of several parameters in TFTP which makes it secure from MITM attack because several public parameters were already shared between the two communicating parties, prior to the communication. In the next stage of our research work, we want to investigate security methods to ensure the privacy during client server communication.

References

1. Mazalan, L.; Hashim, H.; Isa, M.A.M.; and Aziz, N.A. (2011). Attestation with trusted configuration machine. *IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, 570–573.
2. Isa, M.A.M.; Hashim, H.; Manan, J.A.; Mahmod, R.; and Othman, H. (2012). Integrity Verification Architecture (IVA) based security framework for windows operating system. *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 1304–1309.
3. Riemersma, T. (2012). Extending TFTP. Retrieved July 9, 2012, from http://www.compuphase.com/tftp.htm.
4. Lear, E. (2003). RFC 3617 Uniform Resource Identifier (URI) scheme and applicability statement for the Trivial File Transfer Protocol (TFTP). Reterieved October 2003, from https://tools.ietf.org/html/rfc3617.
5. Qiu, S.B.; Yuan, B.; and Zhang, K.L. (2008). Building TFTP server on embedded system. *4th International Conference Wireless Communication Network Mobile Computing*, 1–4.

6. Diffie, W.; and Hellman, M.E. (1976). New directions in cryptography. *IEEE Transactions of Information Theory*, 22(6), 29–40.
7. Yoon, E.J.; and Yoo, K.Y. (2009). An efficient diffie-hellman-MAC key exchange scheme. *Fourth International Conference on Innovative Computing, Information and Control*, 398–400.
8. Coppersmith, D. (1994). The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3), 243-250.
9. Alanazi, H.O.; Zaidan, B.B.; Zaidan, A.A.; Jalab, H.A.; and Shabbir, M. (2010). New comparative study between DES, 3DES and AES within nine factors. *Journal of Computing*, 2(3), 152–157.
10. Singhal, N.; and Raina, J.P.S. (2011). Comparative analysis of AES and RC4 algorithms for better utilization. *International Journal of Computing Trends Technology*, 1(3), 177–181.
11. Prasithsangaree, P.; and Krishnamurthy, P. (2003). Analysis of energy consumption of RC4 and AES algorithms in wireless LANs. *GLOBECOM '03. IEEE Global Telecommunication Conference*, 1445–1449.
12. Fluhrer, A.S.; and Mantin, I. (2001). Weaknesses in the key scheduling algorithm of RC4. *8th Annual International Workshop on Selected Areas in Cryptography*, 1-24.
13. Information, F.; and Standards, P. (2012). Secure Hash Standard (SHS). Retrieved June 3, 2012, from http://dx.doi.org/10.6028/NIST.FIPS.180-4.
14. Singh, K.J.; Manimegalai, R.; and Nadu, T. (2012). A survey on joint compression and encryption techniques for video data. *Journal of Computer Science*, 8(5), 731–736.
15. Razzaque, A.; and Thakur, N.V. (2012). Image compression and encryption : an overview. *International Journal of Engineering Research and Technology*, 1(5), 1–7.
16. Chaudhari, M.; and Saxena. K. (2013). Fast and secure data transmission using symmetric encryption and lossless compression. *International Journal of Computer Science and Mobile Computing*, 2(2), 58–63.
17. Navin, A.H. (2011). A novel approach cryptography by using residue number system. *6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 636-639.
18. Horvat, G.; Zagar, D.; and Martinovic, G. (2013). STFTP: Secure TFTP protocol for embedded multi-agent systems communication. *Advances in Electrical and Computer Engineering*, 13(2), 23-32.
19. Isa, M.A.M.; Mohamed, N.N.; Hashim, H.; Adnan, S.F.S.; Manan, J.A.; and Mahmod, R. (2012). A lightweight and secure TFTP protocol for smart environment. *International Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, 302–306.